TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

BAKALÁŘSKÁ PRÁCE

Liberec 2012 **Jan Hybš**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Serverová část systému pro podporu praktické výuky programování

Server part of the programming training supporting system

Bakalářská práce

Autor: Jan Hybš

Vedoucí práce: doc. Ing. Jiřina Královcová, Ph.D.

V Liberci 18. 5. 2012

Zde bude vloženo originální zadání práce

PROHLÁŠENÍ

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

PODĚKOVÁNÍ

Chtěl bych tímto poděkovat všem, kteří se podíleli na výsledku této Bakalářské práce. Byla to především má rodina, která mi poskytla podporu finanční, psychickou a jinou, a bez které bych nebyl schopen práci dokončit. Velký dík patří mému bratrovi, který svým grafickým umem vdechl život celé práci. Děkuji přátelům, učitelům, studentům a všem dalším, kteří mi pomohli s realizací této práce.

Na závěr bych chtěl poděkovat vedoucí práce doc. Ing. Jiřině Královcové, Ph.D. za poskytnuté konzultace, užitečné rady a za možnost zabývat se touto problematikou.

ABSTRAKT

Tato práce se zabývá vytvořením systému, který je schopen automatizovaně zpracovávat přijatá řešení, což jsou zdrojové kódy vytvořené studenty, na algoritmické úlohy a zároveň poskytnout učitelům výsledky a statistiky odevzdaných řešení a sledovat jejich náročnosti. Další činností systému je kontrola podobnosti přijatých řešení sloužící pro případné odhalení plagiátorství.

Výsledný systém nazvaný „CoDiAna“ (**Co**de **Di**agnostic **Ana**lyzer) byl implementován s využitím technologií Java, Flex, PHP, Python, MySQL a HTML. Vytvořený systém se skládá ze strany klienta a strany serveru. Strana klienta byla částečně realizována v rámci Bakalářského projektu, ale přesto došlo k aktualizaci celé strany klienta. Klientská strana, využívající technologie Flex a HTML, slouží pro komunikaci se stranou serveru a je realizována pomocí grafického uživatelského rozhraní. Strana serveru, využívající technologie Java, PHP, Python a MySQL, zpracovává požadavky od strany serveru. Na základě požadavků poté ukládá potřebná data do MySQL databáze pomocí technologií PHP a Java. Zpracování zdrojových kódů je realizováno pomocí Java aplikace, která prostřednictvím komunikace s MySQL databází, kompiluje a spouští přijatá řešení jazykem Python.

Vhodným návrhem systému byla zajištěna flexibilita a případná další rozšiřitelnost celého systému. Navržený systém zcela podporuje programovací jazyk Java. Systém navíc může podporovat další programovací jazyky. Jejich podpora závisí na prostředcích serveru a připravenosti Java aplikace.

Realizace této práce byla rozdělena na několik částí. Nejprve byla provedena analýza možností použitých prostředků, dále byla aktualizována strana klienta. Při budování strany serveru byl nejprve vypracován PHP server spolu s MySQL databází. Byla tak zajištěna základní funkčnost systému – vytváření a příjem řešení. Na závěr byla navrhnuta a realizována Java aplikace, která doplnila funkčnost systém o zpracování řešení a test podobnosti řešení. Výsledný systém CoDiAna může v předmětech zabývajících se programováním a algoritmizací nahradit nebo alespoň doplnit kontrolu samostatných prací vypracovaných studenty.

KLÍČOVÁ SLOVA

Server klient, programování, analýza zdrojových kódů, Java, webová aplikace

ABSTRACT

This work deals with creating supporting system which is able to automatically process received solutions (i.e. source codes created by students) on algorithmic tasks while providing results and statistics to the teachers and monitor their performance. Another capability of the system is the similarity control of the received solutions serving to the potential plagiarism detection.

Created system called "CoDiAna" (**Co**de **Di**agnostic **Ana**lyzer) was implemented using technologies Java, Flex, PHP, Python, MySQL and HTML. Created system consists of the client and server side. Client side was partially realized within Bachelor project and yet whole side was actualized. Client side – using technologies Flex and HTML – is used for communication with the server side and it is realized by graphic user interface. Server side – using technologies Java, PHP, Python and MySQL – processes received requests from the client side. Server stores necessary data to the MySQL database through technologies PHP and Java based on the requests. Processing of the source codes is realized using Java application which communicates with MySQL database, compiles and executes received solution using programming language Python.

Appropriate system design ensures flexibility and other system potential extensibilities. Suggested system entirely supports programming language Java. Also system can supports other programming languages. Its support depends on server capabilities and readiness of the Java Application.

Realization of this labor was divided into several parts. At first was made analysis of the capabilities of the used technologies then was updated client side. When building server side, PHP server and MySQL database was developed as first. This way ensured partial system functionality – creation and intake of solutions. Java application which was designed and realized at the end complemented system functionality by solutions processing and solutions similarity test. CoDiAna system can be definitely used in subject dealing with programming and algorithmization.

KEYWORDS

Server client, programming, source code analysis, Java, web application

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK, SYMBOLŮ A TERMÍNŮ

- Server

  > *Počítač/počítače nebo počítačové programy poskytující klientovi různé služby.*

- Klient

  > *Uživatel (nebo aplikace) využívající prostředky serveru.*

- Adobe Flash

  > *Vektorový program pro tvorbu dynamických aplikací (především hry, animace, bannery, reklamy a video přehrávače). Při tvorbě interaktivní aplikace je používán programovací jazyk ActionScript.*

- Flash (termín)

  > *Označení pro aplikaci vytvořenou programem Adobe Flash. Výsledná aplikace je tvořena jedním souborem, který má koncovku „swf" nebo „exe".*

- ActionScript

  > *Objektově orientovaný multiplatformní programovací jazyk.*

- Apache Flex

  > *Sada technologií pro tvorbu multiplatformních internetových aplikací založená na technologii Flash. Při tvorbě Flex aplikací je použit programovací jazyk ActionScript a MXML.*

- MXML

  > *Značkovací programovací jazyk popisující grafické rozhraní a veškeré interakce s výslednou aplikací pomocí jazyku ActionScript založený na XML.*

- XML

  > *Značkovací jazyk pro reprezentaci dat.*

- PHP

  > *Skriptovací programovací jazyk na straně serveru.*

- MySQL

  > *Databázový systém, sloužící pro úchovu dat a práci s daty pomocí dotazovacího jazyku (tzv. SQL)*

- Java

  > *Objektově orientovaný multiplatformní interpretovaný programovací jazyk.*

- Python

  > *Objektově orientovaný multiplatformní skriptovací programovací jazyk.*

- HTML

  > *Značkovací jazyk pro tvorbu internetových stránek.*

〉 třída (programování)

> *Předpis objektu v určitém objektově orientovaném programovacím jazyku. Třídy obsahují metody, proměnné a další prvky, usnadňující předpis objektu.*

〉 metoda (programování)

> *Sled příkazů sloužící k realizaci určitého problému. Metoda má vlastní identifikátor a přísluší určité třídě.*

〉 rozhraní (programování)

> *Koncept, který popisuje požadavky abstraktního objektu. Neobsahuje žádná data, ale pouze definuje, jaké náležitosti musí třída (nebo rozhraní) splňovat.*

〉 Garbage Collector

> *Automatický správce paměti běžícího programu, který se snaží nalézt osamocené (již nepoužívané) objekty a následně je zničit a uvolnit tak operační paměť.*

1 Úvod

Tato práce se zabývá návrhem a implementací softwarového systému pro podporu automatizované kontroly a hodnocení samostatných prací studentů v předmětech zabývajících se algoritmizací a programováním, konkrétně v programovacím jazyce Java.

Základní myšlenka systému je založena na správě sady definovaných úloh. Konkrétní úloha je specifikována zadáním, referenčním řešením a kontrolními vstupními a výstupními soubory. Systém by měl umožňovat odevzdávání řešení na úlohy (řešením se rozumí algoritmus v programovacím jazyce Java). Po odevzdání řešení může systém zahájit kontrolní, testovací a hodnotící činnost tak, jak bylo specifikováno v zadání této práce. Za použití kontrolních vstupních souborů může být řešení spuštěno a vygenerovaný výstup porovnán s odpovídajícími kontrolními výstupy. Systém by měl dále monitorovat časovou náročnost odevzdaného řešení. Výsledkem hodnotící činnosti mohou být informace o případných chybách (chybný výstup, syntaktická chyba, překročení časového limitu, …) nebo informace o korektnosti řešení včetně hodnocení. Systém tak může, do jisté míry, zautomatizovat problematiku kontroly a hodnocení studenty vypracovaných a odevzdaných řešení zadaných programovacích úloh.

Kontrola úloh v předmětech zabývajících se algoritmizací a programováním je zpravidla prováděna „ručně“. Ruční kontrola se může různit, ale vesměs je složena z několika kroků. Kontrola začíná postupným načítáním všech řešení. Každé řešení je podrobeno vizuální kontrole za účelem analýzy struktury algoritmu. Dále je řešení spuštěno a otestováno pomocí testovacích vstupů a výstupů. Na základě výsledků běhu programu a struktury algoritmu je vyvozeno hodnocení odevzdaného řešení. Určité části uvedeného ručního zpracování odevzdaných řešení lze automatizovat; mohou být nahrazeny navrženým systémem. Systém pro podporu praktické výuky programování může realizovat podpůrné činnosti spojené se zadáváním úloh a kontrolou řešení. Může dále provádět testování a navrhovat hodnocení řešení. Na druhou stranu, některé prvky hodnocení, jako je například hodnocení použitých struktur a členění kódu, jsou

významně závislé na individuálním názoru hodnotitele a nemohou být tímto systémem plně zohledněny.

Se systémem tak, jak byl výše definován, mohou pracovat především dva typy uživatelů – uživatel v roli „**učitel**“ a uživatel v roli „**student**“. Učitel úlohy vytváří a zadává. Student úlohy vypracovává a odevzdává. Další kroky celého procesu tj. činnosti kontrolní a hodnotící, včetně uživatelského rozhraní se všemi potřebnými ovládacími a vizualizačními funkcemi pro jednotlivé typy uživatelů, jsou pak záležitostí uvedeného softwarového systému.

Tento text popisuje návrh celého systému. Je vysvětlena struktura systému a jeho možnosti. Blíže jsou přiblíženy „**úlohy**“, které jsou v systému klíčové. Části systému jsou blíže popsány a dále jsou vysvětleny vazby mezi nimi. Na závěr textu je prezentováno ukázkové použití vytvořeného nástroje tak, jak bylo realizováno v průběhu zkušebního provozu systému.

2 NÁVRH SYSTÉMU

2.1 ZÁKLADNÍ INFORMACE O SYSTÉMU

Systém pro podporu praktické části programovaní je označován akronymem **CoDiAna** (**Co**de **Di**agnostic **Ana**lyzer). Systém využívá základní síťovou architekturu klient-server a může být ovládán prostřednictvím webového přístupu. Je složen ze dvou hlavních částí – klientské části a serverové části. Klient (uživatel) posílá požadavky na server a dostává od serveru odpovědi.

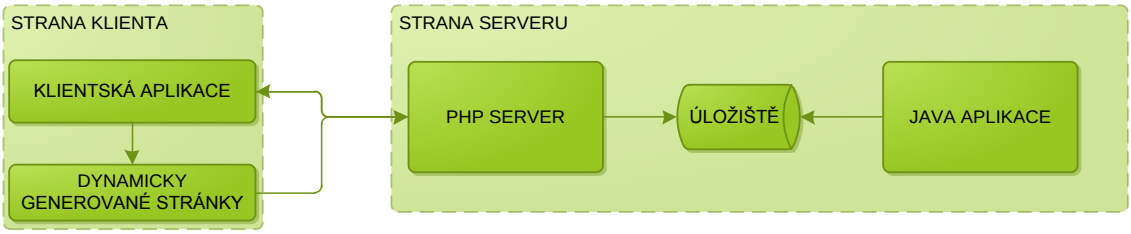
Na serveru jsou uložena všechna data, související se systémem CoDiAna. Pro práci se systémem je nutné načíst webovým prohlížečem adresu serveru. Webová stránka, kterou klient zobrazí, obsahuje dynamickou **klientskou aplikaci**, která komunikuje zpětně se serverem. Dynamická **klientská aplikace** poskytuje prostředky pro práci se systémem a na základě akcí uživatele zasílá požadavky serveru.

2.2 STRUKTURA SYSTÉMU

Klientské i serverové části jsou složeny z dalších sekcí. Každá sekce řeší určitou problematiku systému. Na **Obr.** je zobrazena zjednodušená struktura celého systému. Ukazuje rozložení jednotlivých částí systému.

Klientská aplikace sloužící jako prostředník mezi stranou klienta a stranou serveru, vysílá požadavky na **PHP server**. Ten kontroluje a zpracovává příchozí požadavky a ve většině případů zasáhne do databázového **úložiště**. Po celou dobu zpracovávání požadavků generuje **PHP server** odpověď ve formě XML. V určitých případech může **klientská aplikace** otevřít v prohlížeči novou záložku (například export výsledků). Nově otevřené **dynamicky generované stránky** poté zobrazí požadovaný obsah (PDF soubor, nebo XLS soubor).

Často **PHP server** učiní v **úložišti** změny, na které musí reagovat **Java aplikace**. Tato situace nastává například tehdy, když student odevzdává své řešení. **Java aplikace** průběžně kontroluje **úložiště** a posléze zpracovává odeslané řešení. Po zpracování ať už úspěšném nebo neúspěšném jsou výsledky uloženy do **úložiště**, kde k nim opět může přistoupit **PHP server** a zobrazit výsledky uživateli.



Obr. : Zjednodušená struktura systému

2.3 ROLE V SYSTÉMU

Systém CoDiAna podporuje celkem tři různé role uživatelů. Každá role má k dispozici jinou sadu možných akcí. V zásadě se jedná o rozlišení, zda je role zadavatelem úloh nebo řešitelem úloh. Role uživatelů v systému jsou označeny jako „**učitel**“, „**student**“ a „**skupina**“.

2.3.1 Uživatel v roli učitel

Uživatel v roli učitele je zadavatel úloh a může provádět následující akce:

- vytváření nových úloh,
- úprava a nastavení svých úloh,
- odstranění a archivace svých úloh,
- prohlížení výsledků svých úloh,
- úprava vlastního profilu.

2.3.2 Uživatel v roli student

Uživatel v roli studenta je řešitel úloh a může provádět následující akce:

- zápis na úlohy od různých učitelů,
- odesílání řešení na zapsané úlohy,
- prohlížení svých výsledků (v závislosti na nastavení úlohy může student prohlížet i výsledky ostatních studentů),

- správa svých skupin,
- úprava vlastního profilu.

2.3.3 Uživatel v roli skupina

Skupiny jsou v Systému CoDiAna zvláštním případem studenta. Každá skupina je složena z více studentů, kde mají všichni studenti stejná práva. Výsledky skupiny jsou společné pro všechny studenty uvnitř skupiny. Uživatel v roli skupiny má v systému stejné možnosti jako student. Může navíc upravovat sestavení skupiny.

2.4 ÚLOHY V SYSTÉMU

Klíčovou částí v systému je tzv. „úloha“. Pod tímto pojmem se rozumí algoritmická programovací úloha. V Systému CoDiAna je každá úloha identifikována svým názvem a svým tvůrcem (zadavatelem). Základní údaje každé úlohy jsou následující:

- rozlišení, zda je úloha pouze pro studenty, skupiny nebo je přístupná všem,
- rozmezí, v jakém bude úloha přístupná řešitelům,
- obtížnost úlohy,
- maximální počet odeslaných řešení (pokusů),
- maximální počet přihlášených řešitelů,
- v jakém programovacím jazyce lze úlohu řešit.

Tyto údaje může upravovat pouze učitel, který danou úlohu vytvořil. Ostatní učitelé nevidí úlohy ani výsledky jiných učitelů. U každé úlohy lze navíc nastavit, které informace o odevzdaném řešení mohou řešitelé vidět. Pokud tedy řešitel pošle řešení na některou úlohu, může se počet zobrazených informací různit.

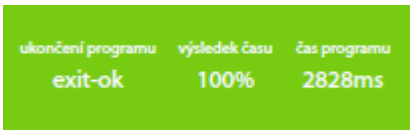
Každé odevzdané řešení musí obsahovat, jakým způsobem byla úloha ukončena. Na **Obr.** lze vidět, jakým způsobem je řešiteli zobrazeno validní ukončení programu.

Obr. ukazuje, jakým způsobem je řešiteli zobrazeno chybné ukončení programu (v tomto případě se jedná o chybu během kompilace programu). Některé chyby v sobě obsahují i zdroj chyby. V tomto případě se jedná o výstup kompilátoru, který obsahuje bližší informace o kompilační chybě.

Každá úloha obsahuje, které položky výsledků budou řešitelům zobrazeny. **Obr.** zobrazuje pouze několik vybraných informací, které jsou řešitelům zobrazeny. Všechny informace o výsledku daného řešení lze vidět na **Obr.** .

Obr. : Validní ukončení programu

Obr. : Chybné ukončení programu



Obr. : Výběr určitých informací o řešení

Obr. : Všechny informace o řešení

2.4.1 Hodnocení úloh

Je nutné, aby každá úloha obsahovala kritéria, podle kterých budou přijatá řešení hodnocena. Tato kritéria musí nastavit učitel u každé své úlohy. Systém CoDiAna umožňuje kontrolu dvou základních parametrů řešení. Jedná se „**čas běhu řešení**“ a „**výstup řešení**“. Oba parametry mají několik režimů kontroly. Tyto režimy umožňují přesnější definici hodnotících kritérií. Každý kontrolovaný parametr se podílí na konečném výsledku řešení. Při kontrole času je na základě doby běhu programu řešení udělena procentuální hodnota pomocí funkce užitku času, viz **Obr.** . Procentuální hodnota výstupu je vypočítána na základě podobnosti s testovacím kontrolním výstupem.

Obr. : Ukázka nastavení hodnotících kritérií z pohledu učitele

2.5 AKCE V SYSTÉMU

Systém CoDiAna podporuje různé akce, které slouží pro práci se systémem. Akce je nadstavba běžného požadavku, který je odesílán na server; obsahuje další klíčové vlastnosti. Každá akce má své unikátní jméno a má definované povinné atributy. Akce v systému CoDiAna jsou následující:

- registrace a přihlášení pro studenty, učitele a skupiny;
- načtení seznamu úloh pro studenta/skupiny nebo učitele;
- načtení detailů uživatele, načtení detailů úlohy;
- vytvoření, základní editace a smazání úlohy;
- potvrzení účtu uživatele, zaslání nového hesla uživateli;
- načtení seznamu studentů/skupin řešící danou úlohu;
- načtení seznamu skupin pro daného studenta;
- schválení nebo opuštění skupiny;
- zapsání do úlohy, odepsání z úlohy, pokračování v úloze;
- nahrání a smazání miniatury uživatele;
- aktivace a deaktivace úlohy, kontrola úlohy;
- nahrání souboru na server, načtení souboru ze serveru;

- nastavení hodnotících kritérií úlohy, nastavení zobrazení výsledků úlohy;
- žádost o test plagiátorství odevzdaných řešení, načtení seznamu podobných řešení;
- zobrazení statistik dané úlohy, export výsledků úlohy.

2.6 PODPORA PROGRAMOVACÍCH JAZYKŮ

Systém CoDiAna byl původně navržen pro podporu programovacího jazyku Java, ale umožnuje i podporu jiných programovacích jazyků. Tato podpora dalších programovacích jazyků závisí na dvou skutečnostech, jsou to:

1. prostředky serveru;
    - Pokud je přidávaný programovací jazyk interpretovaný, je nutné, aby měl server prostředky ke spuštění interpreta daného programovacího jazyka. Kompilované jazyky vyžadují, aby se na serveru nacházely prostředky pro zkompilování a spuštění daného programu (viz kapitola **5.3**).
2. připravenost Java aplikace na straně serveru.
    - V závislosti na daném programovacím jazyku je nutné implementovat kontrolu, kompilaci a spuštění programu v hlavní Java aplikaci. Pokud jsou požadavky splněny, jazyk může být přidán a je u něj zajištěna kontrola časové náročnosti a porovnávání výstupů.
    - Pro kontrolu plagiátorství je nutné definovat šablony, podle kterých je struktura programu analyzována. Tento proces může být komplexní, neboť zahrnuje pochopení syntaxe daného jazyka.

V současné době je plně podporován programovací jazyk Java. Systém je navíc připraven na spouštění řešení v jazyce Python.

3 STRANA KLIENTA

Strana klienta se skládá ze dvou částí. První část je softwarová **klientská aplikace** vytvořená technologií Flex. Druhá část tvoří **dynamicky generované webové stránky**. Převážnou většinu komunikace se serverem provádí klientská aplikace. Tato aplikace je vložena do HTML stránky. Dynamicky generované stránky slouží pouze pro několik akcí. Jedná se zejména o akce, které nemohou být provedeny přímo v klientské aplikaci.

3.1 KLIENTSKÁ APLIKACE

Aplikace je vytvořena v programovacím jazyku ActionScript 3.0 (směsice známých vyšších programovacích jazyků) a využívá technologii Flash. Jelikož je klientská aplikace čistě webová, využívá Framework Flex. Flexové aplikace slouží pro práci s formuláři, zobrazování grafů a statistik, přestože je typickým znakem Flash aplikací je animace a pohyb.

3.1.1 Uživatelské rozhraní

Klientská aplikace má vzhledem k použitému frameworku základní webové ovládací prvky. Mohou to být tlačítka, vstupní pole, zaškrtávací tlačítka apod.

Každý ovládací prvek, který může obsahovat nějaký text, byl rozšířen. Rozšíření umožnilo v dané komponentě podporu multijazyčnosti. Ke všem použitým ovládacím prvkům byly vytvořeny tzv. „skiny“, které nahrazují jejich původní vzhled. Každý „skin“ je zjednodušen, a vede ke zrychlení vykreslování a tedy i zrychlení celého systému. Systém obsahuje další prvky (komponenty), které usnadňují práci se systémem.

3.1.2 Multijazyčnost aplikace

Součástí klientské aplikace je XML slovník. Všechny komponenty, obsahující řetězec, který je v různém jazyce různě interpretovaný, jsou přetíženy a podporují multijazyčnost. Při vytváření každé komponenty je vytvořena slabá reference[1]

[1] Slabá reference je zvláštní případ odkazu na objekt, který je ignorován GC (Garbage Collector)

mezi danou komponentou a statickým objektem slovníku. Tato slabá reference zajistí, že je komponenta odebrána GC (tj. Garbage Collector) přestože existují reference na tuto komponentu. GC ignoruje všechny slabé reference vedoucí k právě odebíranému objektu.

Komponenty, které podporují multijazyčnost, obsahují klíčový identifikační řetězec (dále jen fráze). Podle této klíčové fráze je nalezena hodnota v XML slovníku a zobrazena místo fráze. Aktuálně používaný slovník je dvoujazyčný, ale struktura XML umožňuje přidání dalších jazyků. Záznam ve slovníku vypadá následovně:

```
<klicova-fraze en="english equivalent" cs="český ekvivalent" />
```

Z ukázky lze vidět, že systém slovníku je jednoduchý a rozšiřitelný. Počet frází ve slovníku se blíží k hodnotě 300. Znamená to tedy, že se v aplikaci může objevit téměř 300 českých a anglických frází.

Přepínání slovníků je okamžité. Při přepnutí jazyku vyšle statický objekt slovníků událost o změně slovníku. Každá komponenta svou slabou referencí naslouchá na tuto událost, a když ji zachytí, zažádá tato komponenta o novou hodnotu – komponenta musí být viditelná. Statický objekt ji na základě předložené fráze vrátí ekvivalent buď v českém, nebo anglickém jazyce.

3.1.3 Struktura klientské aplikace

Je nutné, aby se klientská aplikace adaptovala v závislosti na tom, jaký uživatel k ní přistupuje. Existují čtyři základní druhy přístupu k aplikaci. Nejzákladnější je přístup bez přihlášení, kdy je většina funkcionality zamčena. Další přístupy již vyžadují přihlášení a zpřístupňují další akce, v závislosti na roli přihlášeného uživatele.

3.1.3.1 Přístup bez přihlášení

Pokud je k aplikaci přistoupeno bez jakéhokoli přihlášení, je funkcionalita aplikace značně omezena. V tomto režimu se lze pouze přihlásit nebo registrovat (pokud je registrace dovolena). Uživatel bez úspěšné autorizace a autentizace nemůže postoupit dále do systému.

3.1.3.2 Přístup jako učitel

Po přihlášení v roli „**učitel**“, jsou odemčeny téměř všechny akce v systému. Po úspěšném přihlášení může učitel spravovat svůj profil a všechny jím vytvořené úlohy.

Mezi správu úloh patří vytváření nových úloh, editace současných úloh, kontrola výsledků a plagiátorství aktivních úloh a případné mazání dokončených úloh. Učitel může v průběhu spuštěné úlohy pozorovat výsledky (případně je exportovat do různých formátů) a prohlížet řešení od studentů/skupin. Po skončení úlohy (konec úlohy je definován datem a časem), může učitel poslat požadavek k nalezení podobných řešení mezi odevzdanými řešeními a později s těmito výsledky nakládat dle jeho potřeb.

Každé řešení úlohy je vyhodnoceno na základě přiložených souborů od učitele a na základě nastavení hodnotících kritérií. Ke každé úloze je nutné přiložit tři soubory. Jsou to:

- soubor vstupů,
    - *Soubor obsahující data, která budou vstupem všech řešení a na základě kterých bude řešitelův program generovat výstup.*
- podrobné zadání ve formátu PDF,
    - *Soubor, ve kterém je podrobně vysvětlena problematika úlohy. Může obsahovat ukázky vstupů a výstupů pro lepší pochopení zadání.*
- soubor výstupů nebo správné řešení.
    - Soubor výstupů
        - *Soubor obsahující správné výsledky, které by mělo vygenerovat každé odevzdané řešení.*
    - Správné řešení
        - *Jedná se o algoritmus v podporovaném programovacím jazyce, který je spuštěn a generuje soubor výstupů. Pro odvození časového limitu je zde možnost měřit čas přiloženého algoritmu.*

3.1.3.3 Přístup jako student/skupina

Poslední druh přístupu je v roli „**student**“ nebo „**skupina**“. Oba druhy uživatelů mají v systému stejné možnosti. Po úspěšné autentizaci a autorizaci jsou studentovi/skupině zpřístupněny další funkce. Nejdůležitější z nich je správa úloh. Student/skupina může shlédnout podrobné zadání úlohy a případně se přihlásit na danou úlohu – tj. přihlášení mezi řešitele dané úlohy. Pokud je uživatel v roli student/skupina přihlášen mezi řešitele úlohy, může odeslat řešení, které je ihned uloženo do fronty a následně zpracováno.

V závislosti na nastavení viditelnosti ze strany učitele, vidí student/skupina různou kombinaci detailů o jeho řešení. Vždy ale vidí základní údaje, například kód ukončení programu. Jak bylo zmíněno výše, učitel může u dané úlohy povolit zobrazení výsledků ostatních studentů a skupin přihlášených mezi řešitele úlohy. Student má tedy možnost vidět, jak si v konkrétní úloze vedou ostatní studenti/skupiny.

3.1.4 Zabezpečení aplikace

Klientská aplikace je tvořena technologií, která ve finální fázi zkompiluje všechny zdrojové kódy do jednoho souboru, což poskytuje určitou míru bezpečnosti. Tento soubor je uložen na serveru. Soubor je kompilovaný a jeho kód je tedy uzavřený. Existují však licencované programy pro zpětnou dekompilaci. Přesnost dekompilace se však velice různí. S větším projektem je téměř nemožné získat přesné originální zdrojové kódy, ale jen jejich přibližnou podobu.

Nelze se však spoléhat pouze na uzavřenost klientské aplikace, kontrola systému je prováděna i na straně serveru. Strana serveru je pro běžného uživatele nepřístupná – nelze do ní přímo zasahovat. Pokud by byl tedy odeslán podvrhnutý požadavek na server se správným „kontrolním součtem“ (viz dále), ve kterém by se uživatel vydával na jiného uživatele, byl by požadavek odhalen kontrolou na serveru.

Dalším bezpečnostním prvkem je opatření všech požadavků posílaných na server kontrolním součtem. Kontrolní součet je vypočten na základě obsahu požadavku a následně zašifrován pomocí bezpečnostního algoritmu. Na straně serveru je proveden stejný proces jako na straně klienta. Pokud se hodnoty

kontrolních součtů různí, je požadavek považován za nekorektní a je dále ignorován.

Bezpečnost přenášených hesel je zajištěna už na straně klienta, kde klientská strana vytvoří ze zadaného hesla tzv. „haš“[2] , který je poté odeslán přes nezabezpečenou síť. Strana serveru poté znovu vytvoří „haš“ z již obdrženého „haše“, a poté s výsledkem pracuje v závislosti na druhu požadavku.

Další bezpečnostní riziko, které bylo nutno ošetřit, spočívalo v synchronizaci času mezi stranou serveru a stranou klienta. Pokud není čas serveru a klienta správně synchronizován, může dojít k nesrovnalostem. Uživatel má ve většině případů přístup k systémovým prostředkům včetně času. Posunutím času by mohlo bez synchronizace dojít k zpřístupnění akcí, které nejsou zatím povoleny nebo už nejsou povoleny. Finální kontrola požadavků je stanovena i na serveru, ale lze jim předejít již na straně klienta. Kontrola na straně klienta je realizována pomocí odchylky času na straně serveru od času běhu aplikace. Při dotazu na čas v aplikaci je vrácen čas serveru (s možnou minimální odchylkou).

3.2 DYNAMICKY GENEROVANÉ STRÁNKY

Dynamicky generované stránky doplňují funkčnost strany klienta. Pro větší přehlednost a plynulost systému jsou pomocí těchto stránek realizovány požadavky pro stažení souboru. Každý požadavek je opět vytvořen klientskou aplikací. Aplikace vyšle požadavek, tentokrát na operační systém, a žádá o otevření webové adresy. Adresa, kterou vytvořila klientská aplikace, obsahuje všechny potřebné parametry pro otevření daného dokumentu.

V celé aplikaci je celkem pět případů, kdy PHP server generuje dynamickou stránku, jsou to:

1. otevření zadání úlohy;
    - *Student nebo skupina má možnost zobrazit podrobné zadání úlohy.*
    - *Klientská aplikace otevře webovou stránku s PDF zadáním, které vytvořil učitel a přiložil k dané úloze.*

[2] Haš (také „hash”) je výsledek jednosměrné šifry

2. export výsledků do formátu PDF;
    - *Učitel může exportovat výsledky úlohy nebo test plagiátorství do PDF formátu.*
    - *Výsledky mohou být buď pro tisk, kde jsou veškeré grafické prvky vypuštěny, anebo pro webové zobrazení, kde jsou zachovány všechny nebarevné prvky.*
3. export výsledků do formátu XLS;
    - *Učitel může exportovat výsledky úlohy nebo test plagiátorství do XLS formátu.*
4. potvrzení účtu;
    - *Při registraci je nutné správně vyplnit e-mailovou adresu. Na tuto adresu je zaslán potvrzovací e-mail, ve kterém je uveden odkaz pro aktivaci účtu. Navštívením odkazu je účet aktivován.*
5. zaslání nového hesla;
    - *Pokud uživatel zapomene své heslo, má možnost zažádat o nové. Na jeho e-mailovou je poté zaslán e-mail, obsahující odkaz. Po navštívení zaslaného odkazu vygeneruje systém nové náhodné heslo.*
    - *Vygenerované heslo má délku 8 znaků a obsahuje alfanumerické znaky.*

4 STRANA SERVERU

Strana serveru slouží především pro zpracování požadavků ze strany klienta. Vykonává veškeré požadavky a zpětně informuje stranu klienta.

Průběžně si zaznamenává všechny důležité události do úložiště pro další použití. Nezávisle na straně klienta spouští procesy spouštějící odevzdaná řešení od řešitelů a pomocí úložiště poskytuje zpětnou vazbu o běhu úloh. Na vyžádání od učitele kontroluje strana serveru plagiátorství v úlohách analyzováním a porozuměním odevzdaných řešením.

Všechny tyto úkony jsou prováděny pomocí tří základních částí serveru a jejich vzájemnou komunikací. Jedná se o „**úložiště na serveru**“, „**PHP server**“ a „**Java aplikace**“.

4.1 ÚLOŽIŠTĚ NA SERVERU

Nezbytnou a základní součástí serveru je místo/místa, kam jsou ukládány důležité informace. V tomto systému jsou dvě úložiště, kde jsou uloženy klíčové informace.

Prvním z nich je MySQL databáze, která drží klíčové informace o samotné existenci úloh, uživatelů a dalších entit. Tato struktura obsahuje nejen data ale i logiku popisující, jak s daty nakládat. Integritní omezení slouží jako další kontrola dat a udržuje jejich konzistenci.

Druhým uložištěm jsou soubory na serveru v pevně definované stromové struktuře. Soubory na serveru zahrnují především soubory, které slouží pro práci s úlohami. Jedná se například o PDF zadání nebo vstupní a výstupní testovací soubory).

4.1.1 MySQL Databáze

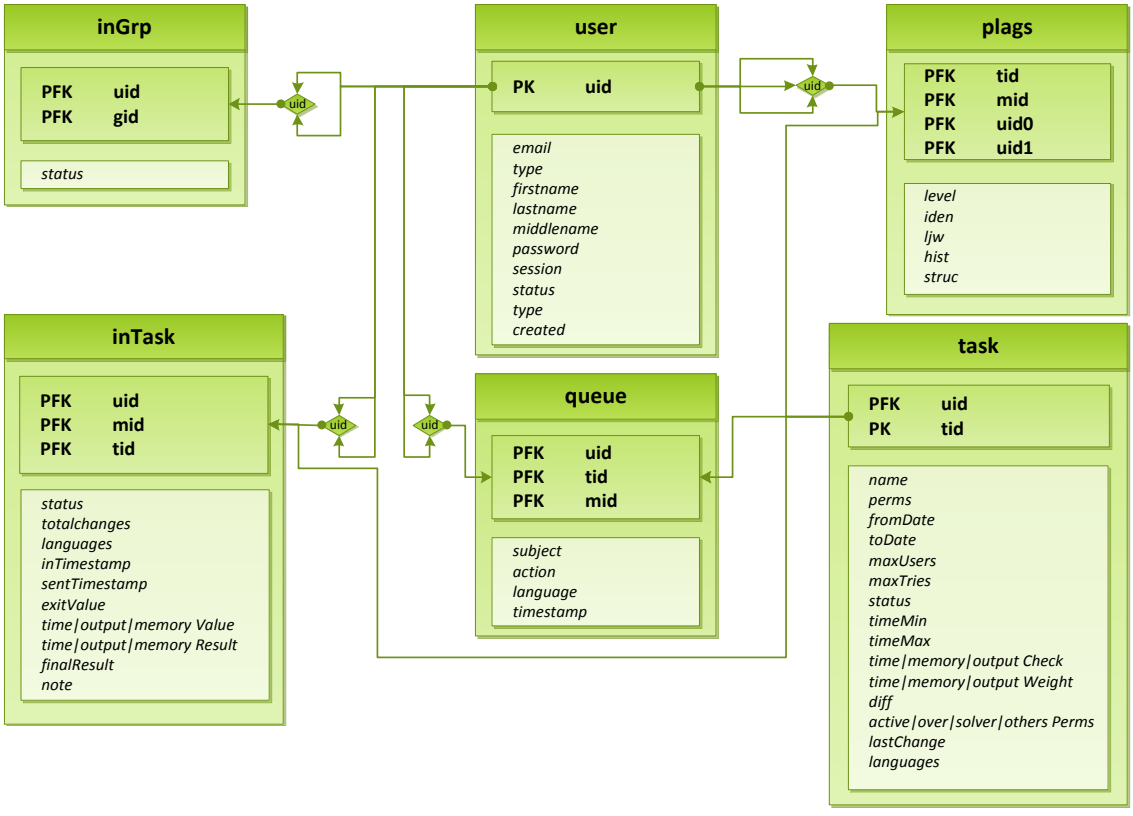
Skrze databázi je do jisté míry zajištěna komunikace mezi PHP serverem a Java aplikací. Po určitém požadavku od strany klienta je do databáze resp. do tabulky „queue“ uložen záznam. Java aplikace tuto hodnotu zaregistruje a učiní potřebné akce.

Celá databáze je složena ze šesti tabulek. Tyto tabulky jsou vzájemně propojené. Při úpravách nebo mazání mohou být změny promítnuty i do ostatních tabulek. Především při mazání vznikají nekorektní data. V tomto systému může uživatel ve výsledku smazat pouze skupiny nebo úlohy, nicméně pokud do systému zasáhne administrátor, může smazat cokoli.

Vnitřní nastavení vztahů relací zajišťuje, že smazáním záznamu v odkazované tabulce budou automaticky smazány všechny záznamy, kde daný záznam figuruje. Je tedy zaručeno, že se v databázi nebudou vyskytovat neúplná data. Pro permanentní ukládání dat slouží soubory na serveru.

Příkladem neúplných dat mohou být:

- student, který je součástí neexistující skupiny,
- zapsání řešitele v neexistující úloze,
- výsledky testu podobnosti neexistující úlohy.



Obr. : Struktura databáze

Každá tabulka má svůj účel a bude blíže popsána pro pochopení funkčnosti databáze.

4.1.1.1 Tabulka „user“

Základní tabulkou databáze je tabulka „user“. Její primární klíč (PK) je odkazován ve všech ostatních tabulkách jako primární cizí klíč (PFK). Vyprázdnění této tabulky by způsobilo vyprázdnění celé databáze. V této tabulce jsou uloženy informace o uživatelích. V celé databázi je každý uživatel identifikován svým PK cože je „UID“. UID může obsahovat několik hodnot. Může to být buď identifikační číslo studenta (`A12345678`) nebo identifikační řetězec učitele (`jmeno.prijmeni`) nebo libovolný název skupiny.

Každý uživatel musí mít unikátní e-mail. Toto omezení se vztahuje na každý druh uživatele. V tabulce se tedy mohou vyskytovat unikátní kombinace složené z e-mailu a druhu uživatele. Ve výsledku to znamená, že stejný e-mail se může vyskytovat maximálně 3× (pro studenta, učitele a skupinu).

Díky skutečnosti, že každá role uživatele stále pochází z tabulky user, je v celé databázi odkazováno na tabulku user více než jednou.

4.1.1.2 Tabulka „inGrp“

Tato tabulka slouží pro umožnění kardinálního vztahu M:N. Uživatel může být v několika skupinách a skupina musí mít minimálně jednoho studenta.

Jelikož student i skupina jsou z pohledu systému pořád uživatelé (viz Tabulka „user“) je dekompozice vztahu řešena pomocí dvou tabulek, kde PFK jsou odkazovány z hlavní tabulky user. Tato tabulka obsahuje ještě pomocný sloupec „status“, kde jsou uloženy informace o tom, zda daný student schválil svoji účast ve skupině.

4.1.1.3 Tabulka „task“

Jak již anglický název napovídá, tato tabulka obsahuje úlohy vytvořené učitelem (PFK). Každá kombinace učitele a názvu úlohy je unikátní. Tímto je zajištěno, že učitelé mohou vytvořit úlohu se stejným názvem, aniž by si museli kontrolovat, zda je název úlohy ještě nepoužitý. Tabulka obsahuje všechna nastavení úlohy, která jsou později použita k vyhodnocení výsledků. Samotné soubory úlohy v databázi nejsou. Jelikož je kardinalita mezi řešiteli a úlohami opět M:N, je nutné použít dekompoziční tabulku.

4.1.1.4 Tabulka „inTask“

Tabulka "inTask" slouží k dekompozici vztahu mezi řešiteli a úlohami. Obsahuje záznam toho, že daný řešitel vypracovává danou úlohu. Tato tabulka obsahuje výsledky řešení řešitele. Při zpracování řešení od řešitele, jsou výsledky uloženy právě v této tabulce.

Hodnoty výsledků řešení obsažené v této tabulce jsou dvojího typu. První typ jsou naměřené hodnoty a druhý typ jsou procentuální hodnoty vzhledem k současným pravidlům úlohy. Poslední sloupec obsahuje finální výsledek řešitele, který je vypočten pomocí procentuálních výsledků a jejich vah. Je možné tuto hodnotu spočítat z již obsažených údajů a údajů uložené v tabulce „task“, nicméně ukázalo se, že je výhodnější vypočítat tuto hodnotu jednou, nežli ji počítat pokaždé s režií spojení tabulek.

4.1.1.5 Tabulka „plags“

Tabulka slouží pro uchování výsledku testů podobnosti řešení. Jeden záznam tabulky obsahuje informace o tom, kteří řešitelé měli podobné výsledky ve které úloze. Dále záznam obsahuje výsledky algoritmů, které sloužily pro odhalení případných plagiátů. Pro přehlednost je v tabulce dále zahrnuta úroveň podobnosti.

4.1.1.6 Tabulka „queue“

Tabulka „queue“ slouží pro zasílání požadavků z PHP serveru do aplikace Java. Každý záznam obsahuje jednoznačný identifikátor dané úlohy a subjektu. Subjektem může být buď učitel, nebo řešitel (tj. student nebo skupina). Dále záznam obsahuje, které operace je nutné provést s danou úlohou (test podobnosti řešení nebo detekce času s generováním výstupu). Důležitým sloupcem je také „timestamp“, podle kterého je určeno pořadí, v jakém budou požadavky zpracovány. Pokud některý z požadavků patří učiteli, je upřednostněn tento požadavek před jinými; tato skutečnost je zajištěna hodnotou sloupce „timetamp“.

4.1.2 Soubory na serveru

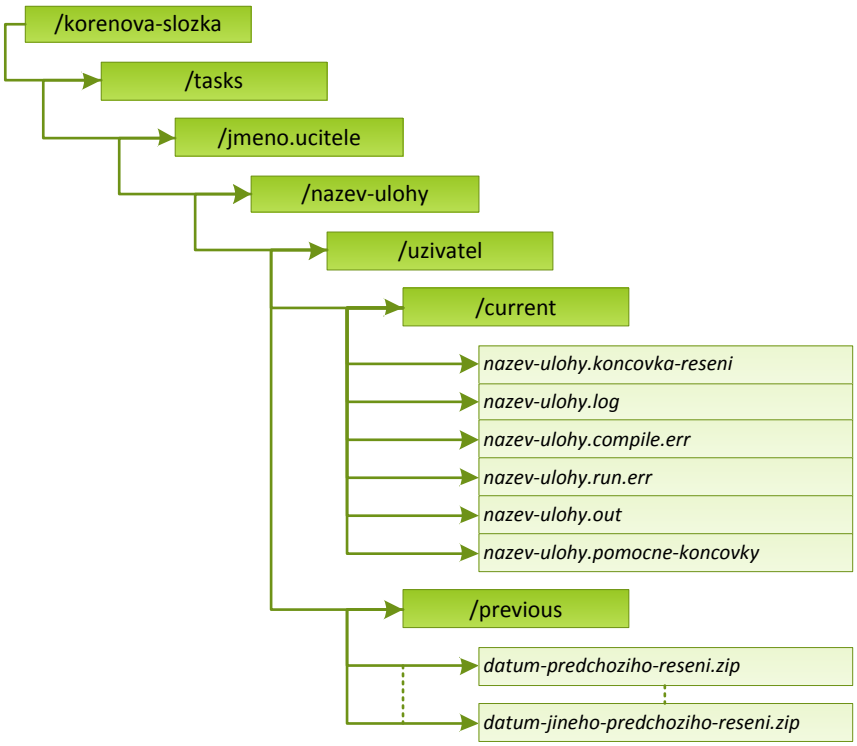
Soubory na serveru jsou použity především pro jejich nutnou existenci na serveru. Odeslané řešení nemusí obsahovat pouze zdrojový kód, nýbrž i archiv obsahující více souborů. V tomto případě je rozbalení archivu provedeno ihned po přijetí.

Java aplikace vyžaduje při spouštění kódu, aby všechny potřebné soubory existovaly, takže uložení zdrojových kódů do databáze není vhodným řešením (soubory by musely být znovu vytvořeny).

Další výhodou je archivování veškerých změn do samostatného archivu pro další použití. Je proto možné prohlížet postupné změny řešitele.

Toto řešení má ovšem i své nevýhody, největší z nich je zabezpečení souboru. Tímto způsobem lze k souboru přistoupit, pokud je ovšem známa cesta. Proto je nutné zakázat vstup do úložiště fyzických souborů.

Struktura souborů řešení na serveru je pevně dána.



Obr. : Stromová struktura úloh na serveru

Na **Obr.** je zobrazena stromovou strukturu na serveru. Všechny úlohy jsou roztříděny podle identifikačního klíče učitele a úlohy. Tato skutečnost zajišťuje větší přehlednost při procházení výsledků přímo na serveru.

Uživatelé mají uložena řešení ve své vlastní složce, jejíž název je tvořen identifikačním klíče uživatele. Pokud klíč obsahuje znaky, které se nesmí vyskytovat v názvu složky, je název převeden na znaky BASE64[3], v opačném případě je název zachován. Klíče studentů a učitelů tuto skutečnost nevyžadují, jména skupin ano. Skupina může mít název libovolný, a proto může obsahovat znaky, které název složky obsahovat nemůže. Tímto je zajištěno, že složka bude vytvořena a systém bude pracovat s jakýmkoli názvem skupiny.

Nevýhodou tohoto řešení, je zhoršená orientace pro správce systému. Na první pohled nelze poznat, jaký název kódované znaky skrývají.

Úlohy, které byly učitelem smazány, jsou archivovány v obdobné struktuře. Zadání úlohy a všechna řešení (bez výstupů) jsou zabalena do jednoho archivu. Velikost tohoto archivu je závislá na počtu odevzdaných řešení, ale je v rozmezí stovek KB až několik MB.

Veškeré požadavky, které přichází na server, mohou být navíc ukládány do logovacích souborů. Tyto soubory obsahují informace o tom, co klient žádal a jak mu na to server odpověděl. Každý log dále obsahuje současný stav serveru.

4.2 PHP SERVER

Částí systému zpracovávající požadavky od klientské strany je PHP server. Požadavky, které server přijímá, jsou převážně od klientské aplikace a jsou typu POST[4]. Dynamicky generované stránky používají pouze požadavky typu GET[5]. Ve výjimečných případech může některý požadavek přenášet první část dat pomocí metody GET a zbylou část pomocí metody POST. Při nahrávání souborů na server přenáší požadavek data opět pomocí metody POST. Server odpovídá ve formátu XML dynamické aplikaci a proudem binárních dat odpovídá dynamickým stránkám.

---

[3] BASE64 je reprezentace dat. Tato reprezentace užívá 64 různých znaků.

[4] GET je metoda HTTP požadavku přenášející data v adrese URL

[5] POST je metoda HTTP požadavku přenášející data v těle požadavku

PHP server vykonává veškerou činnost, která je spojena s komunikací mezi serverem a stranou klienta. Slouží jako komunikační článek mezi uživatelem a požadovanými daty nebo operacemi. Téměř všechny požadavky, které server přijme a akceptuje, pracují s MySQL databází. Navázání spojení s databází je tedy klíčové. Po zpracování (ať už úspěšném či neúspěšném) je vygenerována odpověď. Při exportu dat odešle server hlavičku požadovaného souboru a dále vypíše obsah souboru do prohlížeče.

4.2.1 Struktura PHP serveru

PHP server je naprogramován pomocí objektově orientovaného programování a využívá návrhovou architekturu FCP (Front Controller pattern), kde klient přistupuje pouze k jednomu souboru – přístup k ostatním souborům je zakázán. Tento přístupový soubor poté zpracuje všechny příchozí požadavky.

Server obsahuje několik statických tříd sloužících jako knihovna funkcí. Tyto třídy pomáhají ke zpracování požadavků, vykonávání MySQL skriptu, nebo generování odpovědi pro stranu klienta. Celé zpracování požadavku se skládá z desítek PHP skriptů, které si mohou vzájemně předávat řízení.

4.2.2 Struktura odpovědi

Odpověď pro klientskou aplikaci je vždy ve formátu XML a má pevně danou strukturu. Hlavní uzel obsahuje několik dalších pod-uzlů. V těchto pod-uzlech jsou obsaženy důležité informace, například informace o provedených změnách způsobených MySQL dotazy nebo samotná data z databáze. Strana klienta reaguje na informace obsažené v těchto odpovědích.

Jedním ze základních rozhodovacích prvků je uzel s názvem „operations“, který obsahuje informace o výsledcích provedených MySQL dotazů. Při základní operaci s databází (SELECT, UPDATE, INSERT nebo DELETE) je v uzlu upraven parametr dané operace a obsahuje klíčové informace o úspěchu či neúspěchu operace.

Například při přihlášení je poslán dotaz typu SELECT (tj. výběr). Dotaz vybírá uživatele s příslušnou kombinací jména a hesla. Pokud jsou přihlašovací údaje správné, parametr „select“ pod-uzlu „operations“ bude vypadat následovně:

```
<operations select="ok|1" insert="" update="" delete="" />
```

Klientská aplikace poté detekuje, že se provedla úspěšně operace SELECT u přihlášení, což znamená, že přihlášení proběhlo v pořádku. Pokud by byla chybná kombinace jména a hesla, parametr „select“ bude obsahovat hodnotu „`ok|0`“. Další možností může být chyba v dotazu. Příkladem může být registrace nového uživatele, kde je použit dotaz typu INSERT (tj. vložení). Pokud server příjme žádost o registraci nového uživatele, který již existuje, bude dotaz vyhodnocen s chybou. Hodnota parametru „insert“ bude držet hodnotu „`error|kod-chyby|popis-chyby`“. Klientská aplikace detekuje chybu a informuje uživatele. Celý pod-uzel „operations“ obsahuje několik dalších parametrů, které slouží buď pro zvýšení bezpečnosti, nebo pro upřesnění chyby.

Pokud strana klienta žádá data z databáze, je v XML vytvořen uzel „sql“. Tento uzel obsahuje další strukturované pod-uzly „item“ s výslednými daty. Sloupce dotazu jsou ukládány v dalších pod-uzlech, které obsahují hodnoty řádku. Následuje ukázka XML odpovědi na přihlášení uživatele.

```
<?xml version="1.0" encoding="UTF-8"?>
<x3m>
  <operations time="1329732616" exit="" info=""
   select="ok|1" update="" insert="" delete="" />
  <sql>
    <item>
      <uid>M09000140</uid>
      <firstname>jan</firstname>
      <lastname>hybš</lastname>
      <middlename/>
      <email>jan.hybs@tul.cz</email>
      <type>student</type>
      <status>activated</status>
    </item>
  </sql>
</x3m>
```

Generování odpovědi je prováděno statickou třídou, která obsahuje prostředky pro jednoduché a rychlé generování výstupu.

4.2.3 Zabezpečení serveru

Z bezpečnostních důvodů je každý neznámý požadavek ignorován. Nejsou vyhodnocovány ani požadavky, které nevyhovují pevně daným předpisům požadavku.

Každý požadavek je na serveru definován předpisem. Tento předpis určuje, jaké argumenty musí mít požadavek s daným jménem nebo zda je u toho požadavku nutná kontrola konzistence dat.

Základní kontrola každého požadavku vypadá následovně:

1. kontrola, zda se jedná o validní požadavek,
    - *zdali obsahuje **minimální potřebné** parametry,*
2. kontrola, zda požadavek s tímto jménem existuje,
3. porovnání přijatého kontrolního součtu s právě vypočítaným kontrolním součtem,
    - *klientská aplikace opatří každý požadavek kontrolním součtem,*
4. kontrola, zda je požadavek od správného uživatele,
5. kontrola, zda požadavek obsahuje všechny povinné parametry.

Příkladem předpisu může být například akce „přihlášení“. Tento požadavek je identifikován pod názvem „login“. Požadavek musí obsahovat parametr „heslo“ a parametr „uid“. Pokud požadavek s identifikačním klíčem „login“ nemá požadované parametry, je zastaveno vyhodnocení požadavku a je vrácena odpověď s informací o chybě.

4.2.4 Zpracování požadavků

Pokud proběhne kontrola požadavku bez chyby, je přistoupeno k dalšímu kroku, a to zpracování požadavku. To je realizováno dvěma různými způsoby:

1. vyhodnocení MySQL dotazu,
2. spuštění externího PHP skriptu.

Celý PHP server je do jisté míry flexibilní a to umožňuje automatizované zpracování některých požadavků. Požadavky lze rozdělit podle složitosti na jednoduché a složité. Jednoduchý požadavek je takový, který vykoná SQL dotaz a odevzdá jeho odpověď.

Složitý požadavek je takový, který vyžaduje práci i s jinými zdroji mimo databáze. Požadavek je zpracován odkázáním na jiný PHP soubor, který se o složitější zpracování postará.

Jakékoli vykonání veřejného SQL dotazu [6]automaticky generuje odpověď pro stranu klienta. Soukromé SQL dotazy[7], které slouží pro interní záležitosti systému, nijak neovlivní výstup, přestože jejich výsledky jsou nadále použity.

4.2.4.1 Vykonávání SQL dotazů

Každý SQL dotaz začíná ověřením připojení do databáze. Pokud není navázáno připojení k databázi, je vykonávání ukončeno a v XML výstupu je specifikována chyba. V opačném případě je postupně generován dotaz.

Všechna externí data, která se postupně vkládají do dotazu, jsou zkontrolována. Neošetřená data by mohla způsobit tzv. „**SQL injection**“, kde za pomocí vhodných hodnot externích dat, může dotaz vykonat něco jiného. Jedná se o případy, kde se externí data chovají jako jiná, původně nezamýšlená část dotazu. Přijatá data jsou ošetřena tak, že se před specifické znaky vloží zpětné lomítko. Tento proces se nazývá tzv. „Escapování“.

V některých případech je nutno provést hned několik dotazů na základě pouze jednoho požadavku. Navazují-li dotazy na sebe, je nutné zajistit konzistenci dat. Stav databáze je před dávkou dotazů uložen, takže v případě chyby některé z dotazů, je stav databáze obnoven. Tato skutečnost se nazývá „transakce“. Před vykonáním fronty dotazů je databázi oznámeno, že začíná transakce (START TRANSACTION). V případě chyby dotazu je databáze obnovena (ROLLBACK). Pokud vše proběhne v pořádku, tak je potvrzeno ukončení dávky dotazů (COMMIT).

---

[6] výsledky těchto dotazů jsou zahrnuty v XML odpovědi

[7] výsledky těchto dotazů nejsou v XML odpovědi zahrnuty

4.2.4.2 Spouštění externích PHP skriptů

Komplexnější úkony na straně serveru potřebují více kódu pro své vykonání. Složitější akce, ve kterých se nejedná pouze o SQL dotaz, pracují s dalšími prostředky na serveru. Jedná se o práci se souborovým systémem, zásah do výsledků z SQL dotazu, nebo o práci s globálními proměnnými. V externích skriptech jsou volány především soukromé dotazy. Příkladem Jednoduchého externího skriptu může být například přihlášení. Při přihlášení je nejenom proveden SQL dotaz, ale zároveň jsou uložena klíčová data z výsledku do SESSION[8]. Data v objektu SESSION jsou uložena na serveru a v tomto systému slouží pro ukládání údajů o přihlášeném uživateli.

[8] SESSION je v jazyce PHP superglobální proměnná, obsahující pomocná data uložená na serveru; data nejsou přístupná klientovi.

4.3 JAVA SERVEROVÁ APLIKACE

Poslední částí systému je Java serverová aplikace běžící na pozadí serveru. Java aplikace zajišťuje zpracování přijatých řešení od studentů a test podobnosti přijatých řešení. Aplikace je spuštěna a průběžnou kontrolou zjišťuje, zda není potřeba provést specifické úkony. Operace, které aplikace vykonává, vyžadují práci s pokročilejšími prostředky. Java aplikace rozlišuje dva typu úkonů, jedná se o:

1. spuštění přijatých kódů (detekce časové náročnosti a vygenerování výstupu) a porovnání výstupu s předlohou – „zpracování odevzdaných řešení“;
2. načtení souboru řešitelů dané úlohy a kontrolování podobnosti mezi všemi soubory, mezi všemi třídami a mezi všemi metodami pomocí několika algoritmů – „test podobnosti odevzdaných řešení“.

V případě zpracování odevzdaných řešení se jedná především o práci s vlákny, monitorovacími prostředky, práci s databází a soubory na serveru. Pokud probíhá test podobnosti řešení, nejsou používána vlákna, ale práce s řetězci a analýza kódů.

SQL Dotaz pro získání informací ve frontě v sobě zahrnuje i spojení další tabulky. Java aplikace má takto k dispozici nejenom záznam z fronty, ale i další potřebné informace k spuštění kódu.

Záznam obsahuje:

- identifikační název úlohy;
- identifikační název učitele;
- identifikační název studenta/skupiny, který úlohy odevzdal;
- informace, zda má být vygenerován výstup a detekován čas úlohy;
- použitý programovací jazyk;
- kritéria, podle kterých je vypočítána úspěšnost úlohy.

4.3.1 Struktura aplikace Java

Stejně jako v případě PHP serveru i Java aplikace je vytvořena pomocí objektově orientovaného programování. Objektovost, polymorfismus a dědičnost jsou ve srovnání s PHP serverem používány mnohem častěji (vzhledem k charakteru programovacích jazyků). Hlavní program je rozdělen na několik částí.

Tyto části (v programu nazývané balíčky) jsou logicky uspořádány v závislosti na jejich charakteru. Tímto způsobem je program separován na několik nezávislých částí.

Zjednodušená struktura vypadá takto:

- jádro („core“),
  - *obsahuje klíčové třídy a rozhraní použité v programu.*
- databáze („database“),
  - *obsahuje prostředky pro práci s databází.*
- zpracování řešení („execution“),
  - *kontrola kódů, případná kompilace a následné spuštění řešení;*
  - *na závěr vyhodnocení složitostí.*
- parsování („parsing“),
  - *analýza kódu, seskupování kódu do větších logických bloků (například metody).*
- plagiátorství („derivation“),
  - *kontrola podobnosti kódu pomocí několika algoritmů.*
- doplňky („utils“),
  - *pomocné třídy (převážně statické);*
  - *obsahují například konverzi řetězců, práci se soubory.*

Celá aplikace je postavená tak, aby byla jednoduše rozšiřitelná. Tato skutečnost je zajištěna rozhraními a abstraktními třídami, sloužícími jako předlohy. Aplikace je tak připravena na přidání podpory dalších programovacích jazyků nebo přidání dalších algoritmů pro testování podobnosti.

4.3.2 Běh aplikace

Je doporučováno, aby Java serverová aplikace běžela spolu se serverem pro zajištění plné funkcionality. Pokud by Java aplikace nebyla spuštěná, tak by odevzdaná řešení nebyla zpracována a uživatelé by tak neviděli své výsledky. Jejich odevzdání řešení by způsobilo přidání (nebo upravení) záznamu v tabulce „queue“. Kontrola úloh ve frontě by byla provedena při dalším spuštění Java aplikace.

Po spuštění aplikace jsou zkontrolovány konfigurační soubory. Dále jsou vytvořena pomocná vlákna, sloužící pro periodickou kontrolu databáze a přijetí asynchronních požadavků od správce systému. Pokud je při kontrole databáze nalezen v tabulce „queue“ nějaký záznam, je na základě typu nalezeného záznamu spuštěna obslužná rutina. Záznam může být dvojí, „zpracování odevzdaných řešení“ nebo „test podobnosti odevzdaných řešení“. Po skončení zpracování je záznam odstraněn z tabulky. Celá procedura se opakuje, dokud Java aplikace běží (a není správcem systému pozastavena).

4.3.3 Zpracování odevzdaných řešení

Zpracování kódu studenta (resp. skupiny) a učitele je až na několik odlišností stejné. Je spuštěno ve chvíli, kdy je ze záznamu fronty zřejmé, že se jedná o zpracování kódu. Postup při zpracování je rozložen do několika kroků. Postup je následující:

1. načtení záznamu z tabulky;
2. určení, zda se jedná o zpracování kódu;
    - *Každý záznam obsahuje kód, který jednoznačně identifikuje, co* je nutno *provést.*
3. kontrola, zda je programovací jazyk podporován;
    - *Předtím, než pokročí program ke klíčové části, je provedena prvotní kontrola. Pokud programovací jazyk není podporován, je zpracování kódu ukončeno s chybou indikující nepodporovaný programovací jazyk.*
4. určení cest k použitým souborům;
    - *Jedná se především o soubory vstupu, výstupu chyb, ale i soubory předloh. Potřebné soubory jsou vymazány, aby byla data vždy nově generovaná (mohl by nastat případ, kdy jsou výstupní data porovnána s předchozími výsledky)*
    - *V této části je rozdíl mezi zpracováním kódu pro učitele a studenty/skupiny. Relativní cesty jsou v případě učitele jiné.*
5. kontrola hlavní třídy;
    - *Pro spuštění řešení je nutné, aby nebyla hlavní (spouštěcí) třída v žádném balíčku a byla pojmenována správně. Pokud tomu tak*

není, hlavní třída není jednoznačně identifikovatelná a zpracování kódu je ukončeno s chybou „nenalezena hlavní třída“.

6. příprava pro kompilaci a spuštění;
    - *V této části probíhá naplnění vykonávací třídy soubory a omezujícími podmínkami (např. maximální čas operace).*
7. kompilace kódu;
    - *V závislosti na charakteru programovacího jazyku je provedena kompilace zdrojových kódů. Pokud je kompilace neúspěšná, je zpracování ukončeno s chybou kompilace.*
8. spuštění kódu;
    - *Jsou nastaveny posluchače pro monitorování výsledku úlohy. Následně je úloha několikrát spuštěna. Pokud dojde při běhu souboru k chybě, je zpracování ukončeno s chybou běhu.*
9. kontrola výsledků;
    - *Probíhá porovnání výstupů a výpočet procentuální hodnoty časové náročnosti.*
10. uložení výsledků;
    - *V jakémkoli případě jsou uloženy výsledky do databáze, aby byla poskytnuta zpětná vazba uživateli. V případě učitele, je uložení výsledku provedeno jiným způsobem.*
11. odstranění záznamu z databáze.
    - *Po zpracování úlohy dojde ke smazání záznamu z fronty, aby nedošlo k opětovnému zpracování stejné úlohy.*

4.3.4 Test podobnosti odevzdaných řešení

Kontrolu plagiátorství lze spustit pouze jednou, a to po uzavření úlohy. Učiteli je nabídnuta možnost této kontroly. S větším počtem řešitelů pro danou úlohu může tato kontrola trvat nějakou dobu. Složitost, s jakou roste počet kombinací studentů, je kvadratického řádu. Pokud je na úlohu přihlášeno 5 studentů, je počet kombinací pouze 10. Při 10 studentech je počet kombinací už 45.

Hromadná kontrola souborů dává prostor k optimalizaci. Předpočítání nedynamických vlastností programu může značně usnadnit dobu zpracování kontroly. Postup při kontrole plagiátorství je následující:

1. načtení záznamu z tabulky,
2. určení, zda se jedná o kontrolu plagiátorství,
3. načtení a kategorizace seznamu řešitelů,
    - Dalším SQL dotazem do databáze je načten soubor všech řešitelů. Seznam je rozdělen do skupin dle programovacího jazyka. Probíhá zde i kontrola, zda jsou pro dané podporovací jazyky implementovány předepsané metody. Pokud jazyk není podporován, je řešení ignorováno a kontrola není provedena.
4. předpočítání dat,
    - Předpočítání proběhne pouze za předpokladu, že má daný programovací jazyk implementován všechny potřebné metody. V případě Java jazyku se jedná o parsování všech souborů úlohy.
    - Pokud nemá programovací jazyk definovány potřebné metody, není předpočítání provedeno.
5. porovnání řešení.
    - Pro zajištění co nejpřesnějších výsledků jsou proti sobě porovnány všechny soubory. Uvnitř souborů jsou porovnány proti sobě všechny třídy a uvnitř tříd proti sobě všechny metody. Pouze ty nejlepší výsledky (nejvyšší hodnoty podobnosti) jsou zahrnuty do celkového hodnocení.
    - Porovnávání probíhá na základně čtyř algoritmů. Každý algoritmus řeší určitou problematiku duplicity kódu.
        a) Identické porovnání
        b) Levenshtein-Jaro-Winkler porovnání
        c) Histogramové porovnání
        d) Strukturální porovnání

Každý použitý algoritmus má své silné stránky, ale i své nevýhody a slabá místa. Je určený pro určitou oblast detekování plagiátorství. Pokud jsou všechny algoritmy zkombinovány dohromady, dokáží pokrýt značnou část rozpoznávání plagiátorství a je tedy redukována pravděpodobnost falešně pozitivní nebo falešně negativní klasifikace.

4.3.4.1 Algoritmus Identického porovnání

První algoritmus, který porovnává odevzdaná řešení je kontrola absolutní shodnosti obejktů. Algoritmus proti sobě porovnává dva objekty (například metody). Pokud jsou objekty totožné, podobnost je 1,00 (100 %) v opačném případě není podobnost žádná, tj. 0,00 (0 %).

Po skončení porovnávání všech objektů proti sobě je výsledek vždy mezi 0 % až 100 % kde 100 % znamená, že obě řešení jsou totožné. Tento algoritmus je použit jako první, pokud jsou řešení shodná, není nutno provádět další porovnávání. Pokud je nalezena absolutní shoda, test porovnávání řešení je ukončen – ostatní algoritmy by došly ke stejnému výsledku. Pokud shoda není absolutní, jsou na řešení postupně aplikovány další a nyní už komplexnější algoritmy.

4.3.4.2 Algoritmus Levenshtein-Jaro-Winklerova porovnání

Tento algoritmus je složen ze dvou různých algoritmů. Oba algoritmy řeší podobnost řetězců, avšak žádný z nich není určen pro porovnávání zdrojových kódů. Kombinovaný algoritmus ve výsledku kontroluje pouze podobnost dvou řešení, nijak nebere v potaz význam nebo rozložení kódu.

První algoritmus je Levenshtein algoritmus pro nalezení vzdálenosti mezi dvěma relativně krátkými řetězci. Pracuje tak, že pomocí jednoduchých operací (smazání, vložení a záměna), zkonstruuje z prvního řetězce druhý. Operace vložení a smazání jsou zrcadlové. Co je pro jeden řetězec vložení, je pro druhý řetězec smazání a naopak [1].

Výsledkem algoritmu je vzdálenost mezi řetězci (tj. počet operací nutných pro proměnu z jednoho řetězce do druhého). Máme-li řetězce A a B

```
A = "vzdálenost"
B = "vzdal lenost"
```

Jsou nutné pouze tři úpravy pro získání druhého řetězce z prvního.

1. Záměna písmena „a“ na písmeno „á“ nebo naopak.
2. Smazání mezery „ “ u řetězce A nebo přidání písmena „e“ u řetězce B.
3. Přidání písmena „l“ u řetězce A nebo smazání písmena „l“ u řetězce B.

Druhý algoritmus je Jaro-Winklerova vzdálenost. Na rozdíl od Levenshtein algoritmu je výsledem tohoto algoritmu hodnota od 0 do 1, kde 1 je absolutní podoba. Jaro-Winkler algoritmus hledá počet stejných znaků u obou řetězců a počet přemisťovaných znaků mezi oběma řetězci. Na základě těchto tří hodnot, je průměrem vypočten výsledek. Pokud by algoritmus Jaro-Winkler porovnával stejné řetězce jako v minulém případu, byl by výsledek 88.3 % [2].

Sloučením obou algoritmů, lze úspěšně kontrolovat podobnost zdrojových kódů. Základem je algoritmus Levenshtein, kde nyní nejsou porovnávány znaky, ale řádky. Porovnání znaku bylo v předchozím případě pouze shoda/neshoda. Řádky jsou proti sobě porovnávány pomocí algoritmu Jaro-Winkler. Sloučený algoritmus obsahuje další operaci, která blíže popisuje podobnost dvou řádků. Na základě výsledku algoritmu Jaro-Winkler vzniká nová klasifikace „lehká editace“.

Shoda dvou řádků je určena hranicí shody podobnosti řádků. Hranice pro „shodu“ a „lehkou editaci“ jsou variabilní. Pokud jsou hranice správně nastaveny (kompromisem), může malá změna v jednom řádku vyústit opět ve shodu, což je žádanější. Následuje ukázka porovnání dvou metod.

Tab. : Ukázka porovnání zdrojových kódů

01	`public int op (float p, float q) {`	01	`public int op (float a, float b) {`	Shoda
02	`  double tmp;`			Přidání
03	`  tmp = (p + q) / (p - q);`	02	`  double c = (a + b) / (a - b);`	Editace
04	`  String str = "" + tmp;`	03	`  String s = "" + c;`	Záměna
05	`  int i, result = 0;`	04	`  int j, result = 0;`	Shoda
06	`  for (i = 0; i < str.length (); i++)`	05	`  for (j = 0; j < s.length (); j++)`	Shoda
07	`    if (str.charAt (i) == '0')`	06	`    if (s.charAt (j) == '0')`	Shoda
08	`    result++;`	07	`    result++;`	Shoda
09				Shoda
10	`  return result;`	08	`  return result;`	Shoda
11	`}`	09	`}`	Shoda

Z tabulky lze vidět, že přestože některé řádky nebyly stejné, byly klasifikovány jako shodné. Přejmenování názvů proměnných je tedy do jisté míry neúčinné. Z deseti řádků nejsou dva řádky shodné, tzn., že výsledná známka činí 80 %. Se zvedajícím se počtem přejmenování může klesat nepřesnost algoritmu v závislosti na kontextu.

4.3.4.3 Algoritmus histogramu

Předchozí algoritmus ukázal svou nepřesnost, pokud je na porovnávaný kontext aplikováno přejmenování. U tohoto algoritmu nepřesnost klesá pomaleji. Každý z kódu je rozložen na tzv. „tokeny“. Po seskupení shodných tokenů je vytvořen histogram, který obsahuje klíčový token a počet zastoupení daného tokenu. Oba histogramy jsou následně porovnány. Pro porovnání jsou použity všechny tokeny z obou histogramů. Výsledkem je číslo, které indikuje, kolik tokenů bylo různých. Nevýhodou algoritmu je, že i naprosto rozdílné kódy mohou mít shodný histogram.

Pokud by byly nahrazeny všechny proměnné zcela rozdílnými názvy, tak by algoritmus porovnával už pouze vnitřní stavbu kódů. Výsledek by se tedy zastavil na hranici, která je poměrem mezi systémovou a uživatelskou částí. Výsledná známka při porovnání předchozích metod činila 83 %.

4.3.4.4 Algoritmus porovnání struktury

Všechna řešení, která jsou porovnána, byla předem analyzována a rozdělena na větší logické bloky. Největší bloky jsou třídy, rozhraní a výčtové typy. Nejmenší bloky jsou metody.

Při hledání struktury kódu, je nutné prohloubit toto hledání. Každá metoda obsahuje další prvky. Každá metoda je složená z příkazů (ve většině případů je každý řádek příkazem). Pro uskutečnění porovnání struktury je nutné blíže určit tyto příkazy. Pro identifikování těchto příkazů, je nutná znalost syntaxe daného programovacího jazyka. V programovacím jazyce Java jsou to podmínky, cykly, výjimky a mnohé další, ale v jiných programovacích jazycích to mohou být například „lambda funkce“, „seznamy s porozuměním“ nebo několikanásobná přiřazení.

Struktura jazyku Java obsahuje v této práci přibližně dvacet druhů příkazů, některé lze dále ještě dělit. Postupnou analýzou kódu jsou vytvořeny struktury porovnaných kódů. Nejvhodnější algoritmus pro samotné porovnání je algoritmus **Levenshtein-Jaro-Winklera**. Předností **algoritmu porovnání struktury** je úplná imunnost proti přejmenování proměnných. Algoritmus ignoruje všechny hodnoty a názvy proměnných, takže je zde možnost, že různé algoritmy (co se do názvů a hodnot týče) mohou mít stejný výsledek.

Pokud by byl použit algoritmus struktury na předchozím příkladu, byla by hodnota 90 %,

5 VÝSLEDKY PRÁCE

Doba realizace systému CoDiAna trvala bezmála dva roky. Tato bakalářská práce navazuje na bakalářský ročníkový projekt [3], ve kterém byly navrženy některé moduly. V průběhu vytváření systému bylo vytvořeno několik verzí systému CoDiAna.

Vytváření softwaru nebylo zpočátku optimální. Byly nejprve testovány možnosti použitých technologií a poté vytvářeny různé části systému pro ověření funkčnosti různých modulů. Některé procedury byly opakovány – tvorba systému tak trvala delší dobu, než bylo nutné. Výuka některých předmětů přinesla nový pohled na věc a pomohla jednoznačně určit osnovu vytváření softwaru.

Struktura předchozích navržených systémů nebyla promyšlená do detailů, což vedlo k častým změnám ve stavbě systému. Postupem času nabyté zkušenosti vedly k výraznému zlepšení systému. Navržený systém byl vytvořen tak, aby podporoval všechny požadavky. Výsledná navržená struktura systému umožňuje navíc extra funkcionalitu.

Vytvořený systém byl realizován pomocí:

- čtyř programovacích jazyků (Java, ActionScript, PHP, Python),
- jednoho dotazovacího jazyku (MySQL),
- jednoho značkovacího jazyku (HTML).

Systém je složený přibližně z 30 000 řádků kódu, 1 600 metod a funkcí, a cca 300 tříd a rozhraní.

5.1 TESTOVÁNÍ SYSTÉMU

Velice důležitou částí při vytváření softwaru je testování. Je nezbytné odhalit co nejvíce chyb, před uvolněním systému do ostrého provozu. Jednoduchý software obsahuje zpravidla méně chyb. Testy, které jsou vytvořeny pro testování, by měly pokrýt všechny možnosti testovaného subjektu. Čím je software složitější, tím více chyb může obsahovat. Chyby vznikají zejména spojováním menším částí softwaru. Proces odhalení chyb může být v některých případech velice obtížný, zejména pokud v aplikaci dochází k větvení chodu programu (tzv. vlákna) nebo

komunikaci s jinými prostředky (databáze, klient-server komunikace, jiný software). V těchto případech mohou vznikat nepředvídatelné chyby.

Testování softwaru by mělo odhalit co nejvíce chyb. Pokud v průběhu testování nejsou nalezeny žádné chyby, neznamená to, že software je bezchybný. Ve většině případů to znamená, že bylo testování nedostatečné (nebylo důkladné, bylo špatně navržené).

V průběhu testování systému CoDiAna bylo nalezeno nemalé množství chyb. Neznamená to však, že je systém nekvalitní, nýbrž to znamená, že bylo v systému nalezeno velké množství případů, které by mohly vyvolat chybu. Chyby vznikaly například při spouštění odevzdaných řešení od studentů/skupin nebo při komunikaci klienta se serverem.

5.2 ZKUŠEBNÍ PROVOZ

Před uvolněním systému do ostrého provozu, je užitečné podrobit systém zkušebnímu provozu. Ve zkušebním provozu je systém dostupný malému počtu uživatelů/testerů, kteří pracují se systémem tak, aby prozkoumali všechny jeho možnosti. Ve zkušebním provozu je na systém pohlíženo z různých úhlů. Testeři mohou vidět některé věci jiným způsobem než programátor, což může vést ke zlepšení systému.

Ve zkušebním provozu byl systém CoDiAna dostupný deseti studentům. Autor systému vystupoval ve zkušebním provozu v roli studenta a zároveň v roli učitele, kde vytvořil deset zkušebních úloh a poskytl je testerům. Během několika dní se začaly objevovat výsledky od studentů. Úlohy byly nakonfigurovány tak, že výsledky studentů nebyly anonymní – bylo tedy možné vidět výsledky ostatních řešitelů. Tato skutečnost vyvolala mezi testery soutěživost. Testeři se snažili vytvořit lepší a rychlejší algoritmy a tím získat vyšší procentuální ohodnocení než ostatní.

Zkušební provoz byl velice užitečný a vedl ke zlepšení systému. V systému byly na žádosti testerů dodefinovány skutečnosti, které vedly ke zpřehlednění a zjednodušení systému. Testeři odhalili další chyby – počet nalezených chyb byl nyní v řádu jednotek, což znamená, že předchozí testování systému bylo provedeno důkladně.

Veškerý provoz systému byl v průběhu zkušebního provozu ukládán na server do logovacích souborů. V případě výskytu chyby byly logovací soubory velice užitečné.

5.3 POŽADAVKY SYSTÉMU

Pro provoz systému CoDiAna je nutné mít k dispozici počítačovou stanici, na které bude nainstalována serverová část systému.

Systém CoDiAna vyžaduje několik prostředků serveru a klienta, aby mohl správně fungovat. Některé prostředky serveru jsou klíčové – nelze bez nich zajistit chod systému a jiné pouze obohacují funkcionalitu systému. Povinné prostředky jsou:

1. webový server (pro zpracování požadavků),
    - *Na cílovém serveru musí být naistalován webový server obsahující modul, který podporuje programovací jazyk PHP. Verze PHP musí být minimálně 5.0.*
2. databázový systém (pro uložení dat),
    - *Server musí obsahovat MySQL databázový systém. Je vhodné, aby verze MySQL byla co nejvyšší (např. verze 4.1) přestože jsou dotazy posílané do databáze konstruovány co nejjednodušeji.*
3. úložiště (pro ukládání dat),
    - *je nutné, aby měl server k dispozici dostatek místa pro ukládání souborů (vstupy, výstupy, zadání, …). Velikost volného místa se může různit. V závislosti na různých faktorech (doba běhu systému, počet úloh, …) velikost, kterou vyžaduje systém CoDiAna kolísá, ale rozpětí je odhadováno na jednotky MB až jednotky GB.*
4. JVM (pro zpracování přijatých řešení, test podobnosti řešení).
    - *Server musí mít k dispozici JVM (Java virtual machine) obsahující JRE (Java runtime environment) – běhové prostředí schopné spouštět Java aplikace. Balíček potřebných prostředků pro tvorbu, zpracování a spuštění Java aplikací je JDK. Minimální požadovaná verze je JDK 1.6.*

Nepovinným prostředkem serveru je například interpret programovacího jazyka Python. Pokud je na serveru tento interpret přítomný, může správce systému zpřístupnit programovací jazyk Python jako další možnou alternativu pro vypracování úloh.

Prostředky pro provoz klientské části jsou oproti prostředkům serveru jednodušší a v dnešní době běžně podporované. Jedná se o:

1. webový prohlížeč,
2. přístup k serveru (většinou se jedná o internetové připojení),
3. Flash player plugin pro prohlížeč,
4. program podporující formát PDF.

Podrobnější požadavky systému lze nalézt v „Příručce pro instalaci systému CoDiAna“. Tato příručka je součástí přiloženého CD.

ZÁVĚR

V průběhu řešení této bakalářské práce byl vytvořen komplexní systém podporující výuku předmětů zabývajících se algoritmizací a programováním. Systém plně podporuje zpracování řešení v programovacím jazyku Java. Oproti zadání navíc systém umožňuje přidání dalších programovacích jazyků. Systém byl navržen tak, aby poskytoval co nejvíce prostředků k definování hodnocení algoritmizačních prací. Systém umožňuje rozsáhlé a detailní nastavení každé vytvořené úlohy. Serverová část systému nabízí možnost ukládání vzorových řešení, vstupních a výstupních souborů. Systém dále automatizovaně archivuje všechna řešení odevzdaná řešiteli a poskytuje navíc prostředky k archivaci celé úlohy. Java aplikace, tj. část serverové strany, kompiluje a spouští odevzdaná řešení. Java aplikace umožňuje detekci časové náročnosti a porovnání výstupů spouštěného řešení s předlohou výstupu úlohy. Dále Java aplikace nabízí prostředky pro testování podobnosti odevzdaných řešení. Ve zkušebním provozu byla úspěšnost odhalení plagiátorských řešení přibližně 80 %. Struktura úložiště, především databáze, byla navržena tak, aby umožňovala automatizovaně sledovat statistiky úloh.

Celý systém sestává ze dvou hlavních částí – části klientské a serverové. Strana klienta byla realizována dynamickou klientskou aplikací s uživatelským rozhraním. Klientská aplikace poskytuje prostředky k práci s celým systémem a navíc podporuje multijazyčnost aplikace. Java aplikace na straně serveru obsahuje grafické uživatelské rozhraní, které zjednodušuje práci se systémem na straně serveru. Java aplikace navíc ukládá protokoly o provedených pracích.

V průběhu řešení bakalářské práce byly rovněž vytvořeny příručky, které popisují práci s klientskou aplikací (uživatelská příručka) a postup instalace systému (průvodce instalací serveru). Pro zjednodušení instalace systému byl navíc vytvořen instalátor, který pomocí několika kroků nainstaluje celý systém do zvolené lokace.

V této práci byl realizován softwarový systém, který může do jisté míry nahradit práci učitele při kontrole řešení programovacích úloh vypracovaných

studenty. Systém svými možnostmi může navíc sloužit i k řešení jiných problémů než pouze automatizovat jistý díl práce pedagoga.

Při testování systému se ukázala i další pozitiva. Testování se zúčastnila skupina studentů třetího ročníku Fakulty mechatroniky, informatiky a mezioborových studií. Systémem průběžně zveřejňované výsledky řešení všech studentů vyvolaly u testerů soutěživost, snahu překonat jiného studenta lepším a rychlejším řešením. Využití systému tak může mít na studenty nezanedbatelné motivační účinky.

Při realizaci systému bylo přihlíženo k bezpečnosti a rychlosti systému. Návrh systému je tedy opatřen různými bezpečnostními prvky jak na straně serveru, tak na straně klienta. Zrychlení systému bylo docíleno redukcí přenášených dat během komunikace mezi klientem a serverem.

Vytvořený systém pro podporu praktické výuky v předmětech zabývajících se programováním tak, jak byl vytvořen v průběhu řešení bakalářské práce, je funkční a může být od příštího akademického roku bez dalších doplnění použit v předmětech programování v jazyce Java.

Tato práce, přestože dokončená, může být dále vhodně rozšířena. Oblast, kterou se tato práce zabývá, je široká; lze tedy tento systém dále vyvíjet. Systém může být obohacen například přesnějším nastavením úloh, kde lze specifikovat knihovny třetích stran nebo povolené prostředky u řešení. Systém CoDiAna může být upraven tak, aby podporoval modularitu – instalované moduly by poté dynamicky rozšiřovaly funkčnost systému, bez nutnosti zásahu do systému samotného. Systém může být například rozšířen o inteligentnější rozpoznávání plagiátorských řešení s využitím genetických algoritmů.

SEZNAM POUŽITÉ LITERATURY

[1] M. Gilleland, „Levenshtein Distance, in Three Flavors,“ [Online]. http://www.merriampark.com/ld.htm. [Přístup získán 18 Leden 2012].

[2] W. E. Winkler, „String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage,“ 1990. [Online]. http://www.amstat.org/sections/srms/Proceedings/papers/1990_056.pdf. [Přístup získán 11 Leden 2011].

[3] J. Hybš, „Software pro podporu praktické výuky z programování,“ Liberec, 2011.

SEZNAM PŘÍLOH

PŘÍLOHA : OBSAH PŘILOŽENÉHO CD

Přiložené CD obsahuje tři složky. V každé složce je jeden nebo více souborů; soubory budou dále popsány. Složky, které CD obsahuje, jsou:

- Složka *dokumentace* obsahující:
  - elektronickou podobu této práce.
- Složka *příručky* obsahující:
  - příručku pro použití systému CoDiAna,
  - příručku pro instalaci systému CoDiAna.
- Složka *software* obsahující:
  - instalační aplikaci systému CoDiAna.

Elektronická podoba práce

Ve složce *dokumentace* je uložena elektronická podoba této práce v několika formátech. Jsou to formáty *pdf*, *docx* a *doc*.

Příručka pro použití systému CoDiAna

Ve složce *příručky* je uložena „příručka pro použití systému CoDiAna“ ve formátech *pdf*, *pptx* a *ppt*.

Příručka na úvod popisuje ovládání systému a prostředí systému. Příručka v další části popisuje registraci v systému a jednotlivé role systému. Dále příručka vysvětluje, jaké jsou možnosti uživatelů v roli student/skupina a učitel. Uživateli v roli student/skupina je vysvětleno, jak může spravovat jeho úlohy. Důležitou kapitolou je forma odevzdávání řešení úloh.

Uživateli v roli učitel je ukázána správa svých úloh a dále detailně popsána tvorba nové úlohy. V příručce je dále zahrnuto, jakými fázemi každá úloha prochází a co je v dané fázi realizováno.

Celá příručka obsahuje kromě textu i ilustrace, které slouží k názornějšímu vysvětlení dané problematiky.

Příručka pro instalaci systému CoDiAna

Ve složce *příručky* je uložena „příručka pro instalaci systému CoDiAna“ ve formátech *pdf*, *pptx* a *ppt*.

Příručka v úvodu popisuje, jaké požadavky musí server splňovat, aby mohl být systém CoDiAna nainstalován a spuštěn. Požadavky jsou děleny na hardwarové a softwarové. Každý požadavek je příručce podrobněji vysvětlen a je popsán účel jednotlivých požadavků. U softwarových požadavků je v příručce nabídnuta alternativa pro tento software spolu s odkazem pro jeho stažení.

V příručce je dále popsána instalace systému. Instalační program je rozdělen do několika kroků. Každý krok je v příručce popsán a ukázán na ilustracích.

Instalační aplikace systému CoDiAna

Ve složce *software* je uložena instalační aplikace systému CoDiAna. Instalátor je uložen pouze v kompresním formátu *jar* (slouží pro programovací jazyk Java).

Instalátor slouží pro instalaci systému na server. Proces instalace je rozdělen na několik kroků, kde uživatel, který instaluje systém CoDiAna, vyplní potřebné údaje.

Instalátor má sobě zabudovány všechny potřebné věci k úspěšné instalaci systému.